(51) International Patent Classification [7] : G06T	A2	(11) International Publication Number: **WO 00/11602**
		(43) International Publication Date: 2 March 2000 (02.03.00)

(21) International Application Number: PCT/US99/19192

(22) International Filing Date: 20 August 1999 (20.08.99)

(30) Priority Data:
60/097,336   20 August 1998 (20.08.98)   US

(71) Applicant: RAYCER, INC. [US/US]; 2585 East Bayshore Frontage Road, Palo Alto, CA 94303 (US).

(72) Inventors: DULUK, Jerôme, F., Jr.; 950 North California Drive, Palo Alto, CA 94304 (US). HESSEL, Richard, E.; 3225 Flemington Court, Pleasanton, CA 94588 (US). GRASS, Joseph, P.; 357 Lennox Avenue, Menlo Park, CA 94025 (US). RASHID, Abbas; 34369 Eucalyptus Terrace, Fremont, CA 94555-1982 (US). HONG, Bo; 2325 Oak Flat Road, San Jose, CA 95131 (US). MAMMEN, Abraham; 2780 Lylewood Drive, Pleasanton, CA 94588 (US).

(74) Agents: CASERZA, Steven, F. et al.; Flehr Hohbach Test Albritton & Herbert LLP, Suite 3400, 4 Embarcadero Center, San Francisco, CA 94111-4187 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR GENERATING TEXTURE
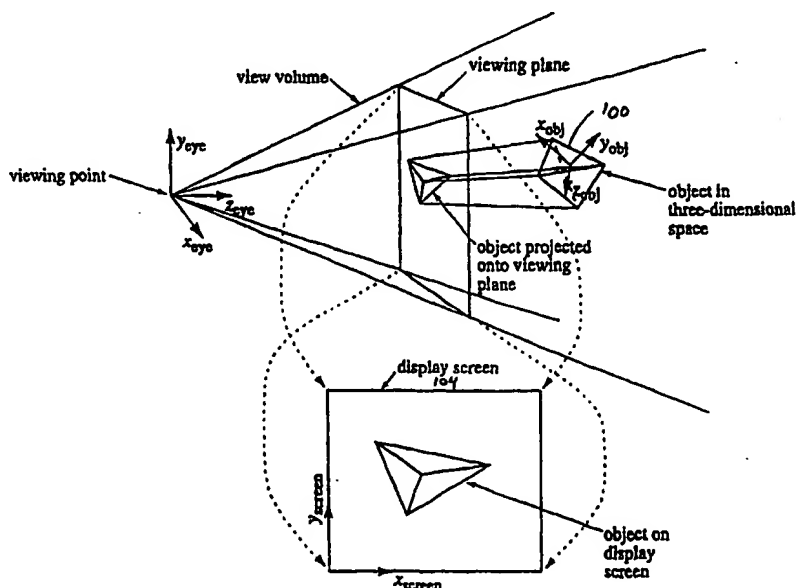
(57) Abstract

A deferred graphics pipeline processor comprising a texture unit and a texture memory associated with the texture unit. The texture unit applies texture maps stored in the texture memory, to pixel fragments. The textures are MIP-mapped and comprise a series of texture maps at different levels of detail, each map representing the appearance of the texture at a given distance from an eye point. The texture unit performs tri-linear interpolation from the texture maps to produce a texture value for a given pixel fragment that approximates the correct level of detail. The texture memory has texture data stored and accessed in a manner which reduces memory access conflicts and thus improves throughput of said texture unit.

Method and Apparatus for Generating Texture

5

Related Applications

This application claims the benefit of U.S. Provisional Application Serial No.

10 60/097,336 entitled Graphics Processor with Deferred Shading filed August 20, 1998 is hereby incorporated by reference.

This application is also related to the following U.S. Patent Applications, each of which are incorporated herein by reference:

Serial No. 09/213,990, filed 17 December 1998, entitled HOW TO DO

15 TANGENT SPACE LIGHTING IN A DEFERRED SHADING ARCHITECTURE (Atty. Doc. No. A-66397);

Serial No. _____, filed _____, entitled APPARATUS AND METHOD FOR PERFORMING SETUP OPERATIONS IN A 3-D GRAPHICS PIPELINE USING UNIFIED PRIMITIVE DESCRIPTORS (Atty. Doc. No. A-

20 66382);

Serial No. _____, filed _____, entitled POST-FILE SORTING SETUP (Atty. Doc. No. A-66383);

Serial No. _____, filed _____, entitled TILE RELATIVE Y-VALUES AND SCREEN RELATIVE X-VALUES (Atty. Doc. No. A-66384);

25 Serial No. _____, filed _____, entitled SYSTEM, APARATUS AND METHOD FOR SPATIALLY SORTING IMAGE DATA IN A THREE-DIMENSIONAL GRAPHICS PIPELINE (Atty. Doc. No. A-66380);

Serial No. _____, filed _____, entitled SYSTEM, APPARATUS AND METHOD FOR GENERATING GUARANTEED CONSERVATIVE

30 MEMORY ESTIMATE FOR SORTING OBJECT GEOMETRY IN A THREE-DIMENSIONAL GRAPHICS PIPELINE (Atty. Doc. No. A-66381);

Serial No. _____, filed _____, entitled SYSTEM,

Serial No. _____, filed _____, entitled METHOD AND APPARATUS FOR PERFORMING CONSERVATIVE HIDDEN SURFACE REMOVAL IN A GRAPHICS PROCESSOR WITH DEFERRED SHADING (Attorney Doc. No. A-66386);

Serial No. _____, filed _____, entitled DEFERRED SHADING GRAPHICS PIPELINE PROCESSOR HAVING ADVANCED FEATURES (Atty. Doc. No. A-66364)

APPARATUS AND METHOD FOR BALANCING RENDERING RESOURCES IN A THREE-DIMENSIONAL GRAPHICS PIPELINE (Atty. Doc. No. A-66379);

Serial No. _____, filed _____, entitled GRAPHICS PROCESSOR WITH PIPELINE STATE STORAGE AND RETRIEVAL (Atty.
Doc. No. A-66378);

Serial No. _____, filed _____, entitled METHOD AND APPARATUS FOR GENERATING TEXTURE (Atty. Doc. No. A-66398);

Serial No. _____, filed _____, entitled APPARATUS AND METHOD FOR GEOMETRY OPERATIONS IN A 3D GRAPHICS PIPELINE (Atty. Doc. No. A-66373);

Serial No. _____, filed _____, entitled APPARATUS AND METHOD FOR FRAGMENT OPERATIONS IN A 3D GRAPHICS PIPELINE (Atty. Doc. No. A-66399); and

Serial No. _____, filed _____, entitled DEFERRED SHADING GRAPHICS PIPELINE PROCESSOR (Atty. Doc. No. A-66360).

Field

This invention relates to computing systems generally, to three-dimensional computer graphics, more particularly to structure and method for generating texture in a three-dimensional graphics processor implementing deferred shading and other enhanced features.

Background

Three-dimensional Computer Graphics

Computer graphics is the art and science of generating pictures with a computer. Generation of pictures, or images, is commonly called rendering. Generally, in three-dimensional (3D) computer graphics, geometry that represents surfaces (or volumes) of objects in a scene is translated into pixels stored in a frame buffer, and then displayed on a display device. Real-time display devices, such as CRTs or LCDs used as computer monitors, refresh the display by continuously displaying the image over and over. This refresh usually occurs row-by-row, where each row is called a raster line or scan line. In this document, raster lines are

numbered from bottom to top, but are displayed in order from top to bottom.

In a 3D animation, a sequence of images is displayed, giving the appearance of motion in three-dimensional space. Interactive 3D computer graphics allows a
5   user to change his viewpoint or change the geometry in real-time, thereby requiring the rendering system to create new images on-the-fly in real-time. Therefore, real-time performance in color, with high quality imagery, is very important.

In 3D computer graphics, each renderable object generally has its own local
10 object coordinate system, and therefore needs to be translated (or transformed) from object coordinates to pixel display coordinates. Conceptually, this is a 4-step process: 1) translation from object coordinates to world coordinates, which is the coordinate system for the entire scene; 2) translation from world coordinates to eye coordinates, based on the viewing point of the scene; 3) translation from eye
15 coordinates to perspective translated eye coordinates, where perspective scaling (farther objects appear smaller) has been performed; and 4) translation from perspective translated eye coordinates to pixel coordinates, also called screen coordinates. Screen coordinates are points in three-dimensional space, and can be in either screen-precision (i.e., pixels) or object-precision (high precision numbers,
20 usually floating-point), as described later. These translation steps can be compressed into one or two steps by precomputing appropriate translation matrices before any translation occurs. Once the geometry is in screen coordinates, it is broken into a set of pixel color values (that is "rasterized") that are stored into the frame buffer. Many techniques are used for generating pixel color values, including Gouraud shading,
25 Phong shading, and texture mapping.

A summary of the prior art rendering process can be found in:
"Fundamentals of Three-dimensional Computer Graphics", by Watt, Chapter 5: The Rendering Process, pages 97 to 113, published by Addison-Wesley Publishing
30 Company, Reading, Massachusetts, 1989, reprinted 1991, ISBN 0-201-15442-0 (hereinafter referred to as the Watt Reference).

Figure 1 shows a three-dimensional object 100, a tetrahedron, with its own coordinate axes $(x_{obj}, y_{obj}, z_{obj})$. The three-dimensional object is translated, scaled, and placed in the viewing point's coordinate system based on $(x_{eye}, y_{eye}, z_{eye})$. The object is projected onto the viewing plane, thereby correcting for perspective. At this point,

5 the object appears to have become two-dimensional; however, in accordance with the present invention, the object's z-coordinates are preserved so they can be used later by hidden surface removal techniques. The object is finally translated to screen coordinates, based on $(x_{screen}, y_{screen}, z_{screen})$, where z_{screen} is going perpendicularly into the page. Points on the object now have their x and y coordinates described by pixel

10 location (and fractions thereof) within the display screen and their z coordinates in a scaled version of distance from the viewing point.

Because many different portions of geometry can affect the same pixel, the geometry representing the surfaces closest to the scene viewing point must be

15 determined. Thus, in accordance with the present invention for each pixel, the visible surfaces within the volume subtended by the pixel's area determine the pixel color value, while hidden surfaces are prevented from affecting the pixel. Non-opaque surfaces closer to the viewing point than the closest opaque surface (or surfaces, if an edge of geometry crosses the pixel area) affect the pixel color value,

20 while all other non-opaque surfaces are discarded. In this document, the term "occluded" is used to describe geometry which is hidden by other non-opaque geometry.

Many techniques have been developed to perform visible surface

25 determination, and a survey of these techniques are incorporated herein by reference to: "Computer Graphics: Principles and Practice", by Foley, van Dam, Feiner, and Hughes, Chapter 15: Visible-Surface Determination, pages 649 to 720, 2nd edition published by Addison-Wesley Publishing Company, Reading, Massachusetts, 1990, reprinted with corrections 1991, ISBN0-201-12110-7 (hereinafter referred to as the

30 Foley Reference). In the Foley Reference, on page 650, the terms "image-precision" and "object-precision" are defined: "Image-precision algorithms are typically

- 5 -

performed at the resolution of the display device, and determine the visibility at each pixel. Object-precision algorithms are performed at the precision with which each object is defined, and determine the visibility of each object."

5   As a rendering process proceeds, most prior art renderers must compute the color value of a given screen pixel multiple times because multiple surfaces intersect the volume subtended by the pixel. The average number of times a pixel needs to be rendered, for a particular scene, is called the depth complexity of the scene. Simple scenes have a depth complexity near unity, while complex scenes can have a depth

10   complexity perhaps within the range of ten to twenty, complexity of ten, 90% of the computation is wasted on hidden pixels. This wasted computation is typical of hardware renderers that use the simple Z-buffer technique (discussed later herein), generally chosen because it is easily built in hardware. Methods more complicated than the Z Buffer technique have heretofore generally been too complex to build in a

15   cost-effective manner. An important feature of the method and apparatus invention presented here is the avoidance of this wasted computation by eliminating hidden portions of geometry before they are rasterized, while still being simple enough to build in cost-effective hardware.

20   When a point on a surface (frequently a polygon vertex) is translated to screen coordinates, the point has three coordinates: 1) the x-coordinate in pixel units (generally including a fraction); 2) the y-coordinate in pixel units (generally including a fraction); and 3) the z-coordinate of the point in either eye coordinates, distance from the virtual screen, or some other coordinate system which preserves

25   the relative distance of surfaces from the viewing point. In this document, positive z-coordinate values are used for the "look direction" from the viewing point, and smaller positive values indicate a position closer to the viewing point.

30   When a surface is approximated by a set of planar polygons, the vertices of each polygon are translated to screen coordinates. For points in or on the polygon (other than the vertices), the screen coordinates are interpolated from the coordinates

- 6 -

of vertices, typically by the processes of edge walking and span interpolation. Thus, a z-coordinate value is generally included in each pixel value (along with the color value) as geometry is rendered.

5   Polygons are used in 3D graphics to define the shape of objects. Texture mapping is a technique for simulating surface textures by coloring polygons with detailed images. Typically, a single texture map will cover an entire object that consists of many polygons. A texture map consists of one or more rectangular arrays of Red-Green-Blue-Alpha (RGBA) color, with alpha being the percentage of

10   translucency. Texture coordinates for each vertices of a polygon are determined. These coordinates are interpolated for each geometry component, the texture values are looked up in the texture map and the color is assigned to the fragment.

       Objects appear smaller when they are farther from the viewer. Therefore,

15   texture maps must be scaled so that the texture pattern appears the same size relative to the object being textured. To avoid scaling and filtering a texture image for each fragment, a series of pre-filtered texture maps, called mipmaps are used. Each texture has a group of associated mipmaps. Each mipmap, also called a level of detail (LOD), is formed of an n x m array of Texture elements (texels), where n and

20   m are powers of 2. Each texel comprises an R, G, B, and A component. Typically each successive LOD has a power of 2 lower resolution than the previous LOD, and thus a cascading series of smaller, prefiltered images are provided, rather than requiring such computations to be performed in real-time. For example, LOD 0 may be a 512 x 512 array, and LOD 9 is 1 x 1 array.

25

       Exact texture coordinates and LOD are typically computed for a sample pixel. The texel values surrounding these texture coordinates are then interpolated to generate texture values for the sample. In bilinear interpolation, the prestored LOD array closest to the computed LOD value is selected, and the values of the four

30   texels in the array nearest to the texture coordinates are interpolated to generate texture values for a sample. In trilinear interpolation, the four texels closest to the

- 7 -

texture coordinates in the prestored LOD arrays above and below the computed

LOD are used to generate the texture values for a sample. For

5   example, if an LOD value of 3.2 is computed then texels from LOD array 3 and

LOD array 4 are used for trilinear interpolation. Trilinear interpolation thus

requires eight texels per sample, which makes high memory bandwidth a critical

component to efficient image rendering.

10

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent

from the following detailed description and appended claims when taken in

conjunction with the drawings, in which:

15

Fig. 1   depicts a three dimensional object and its image on a display screen;

Fig. 2   is a block diagram of one embodiment of a texture pipeline constructed in

accordance with the present invention;

Fig. 3   depicts relations between coordinate systems with respect to graphic

20   images;

Fig. 4a   is a block diagram depicting one embodiment of a texel prefetch buffer

constructed in accordance with the teachings of this invention;

Fig. 4b   is a block diagram depicting texture buffer tag blocks and memory queues

associates with the texel prefetch buffer of Fig. 4a;

25   Fig. 5   is a diagram depicting texture memory organized into a plurality of

channels, each channel containing a plurality of texture memory devices;

Figs. 6a and 6b illustrate a spatially coherent texel mapping for texture memory in

accordance with one embodiment of this invention;

Fig. 6c   depicts address mapping used in one embodiment of this invention;

30   Fig. 7   illustrates a super block of a texture map that is mapped using one

embodiment of the present invention;

- 8 -

Fig. 8   shows a dualoct numbering pattern within each sector in accordance with
         one embodiment of this invention;

Fig. 9   is texture tile address structure which serves as a tag for a texel prefetch
         buffer in accordance with one embodiment of this invention;

5   Fig. 10   is a pointer look-up translation tag block used as a pointer to base address
             within texture memory for the start of the desired texture/LOD in
             accordance of one embodiment of this invention;

Fig. 11   is one embodiment of a physical mapping of texture memory address;

Fig. 12   is a diagram depicting address reconfigurations and process with respect to
10   Figs. 6c, 9, 10, and 11; and

Figs. 13a and 13b are block diagrams depicting one embodiment of a re-order system
in   accordance of the present invention.

DETAILED DESCRIPTION

15   The invention is directed to a new graphics processor and method and
encompasses numerous substructures including specialized subsystems,
subprocessors, devices, architectures, and corresponding procedures. Embodiments
of the invention may include one or more of deferred shading, a tiled frame buffer,
and multiple-stage hidden surface removal processing, as well as other structures
20   and/or procedures. In this document, this graphics processor of this invention is
referred to as the DSGP (for Deferred Shading Graphics Processor), and the
associated pipeline is referred to as the "DSGP pipeline", or simply "the pipeline".

         This present invention includes numerous embodiments of the DSGP
25   pipeline. Embodiments of the present invention are designed to provide high-
performance 3D graphics with Phong shading, subpixel anti-aliasing, and texture-
and bump-mapping in hardware. The DSGP pipeline provides these sophisticated
features without sacrificing performance.

30   The DSGP pipeline can be connected to a computer via a variety of possible
interfaces, including but not limited to for example, an Advanced Graphics Port

(AGP) and/or a PCI bus interface, amongst the possible interface choices. VGA and video output are generally also included. Embodiments of the invention supports both OpenGL and Direct3D Application Program Interfaces (APIs). The OpenGL specification, entitled "The OpenGL Graphics System: A Specification

5   (Version 1.2)" by Mark Segal and Kurt Akeley, edited by Jon Leech, is included incorporated by reference.

Several exemplary embodiments or versions of a Deferred Shading Graphics Pipeline are described here, and embodiments having various combinations of

10 features may be implemented. Additionally, features of the invention may be implemented independently of other features, and need not be used exclusively in Graphics Pipelines which perform shading in a deferred manner.

Tiles, Stamps, Samples, and Fragments

15   Each frame (also called a scene or user frame) of 3D graphics primitives is rendered into a 3D window on the display screen. The pipeline renders *primitives*, and the invention is described relative to a set of renderable primitives that include: 1) triangles, 2) lines, and 3) points. Polygons with more than three vertices are divided into triangles in the Geometry block, but the DSGP pipeline could be easily

20 modified to render quadrilaterals or polygons with more sides. Therefore, since the pipeline can render any polygon once it is broken up into triangles, the inventive renderer effectively renders any polygon primitive. A window consists of a rectangular grid of pixels, and the window is divided into tiles (hereinafter tiles are assumed to be 16x16 pixels, but could be any size). If tiles are not used, then the

25 window is considered to be one tile. Each tile is further divided into stamps (hereinafter stamps are assumed to be 2x2 pixels, thereby resulting in 64 stamps per tile, but stamps could be any size within a tile). Each pixel includes one or more samples, where each sample has its own color value and z-value (hereinafter, pixels are assumed to include four samples, but any number could be used). A fragment is

30 the collection of samples covered by a primitive within a particular pixel. The term "fragment" is also used to describe the collection of visible samples within a

particular primitive and a particular pixel.

Deferred Shading

In ordinary Z-buffer rendering, the renderer calculates the color value (RGB

5   or RGBA) and z value for each pixel of each primitive, then compares the z value of

the new pixel with the current z value in the Z-buffer. If the z value comparison

indicates the new pixel is "in front of" the existing pixel in the frame buffer, the new

pixel overwrites the old one; otherwise, the new pixel is thrown away.

10   Z-buffer rendering works well and requires no elaborate hardware.

However, it typically results in a great deal of wasted processing effort if the scene

contains many hidden surfaces. In complex scenes, the renderer may calculate color

values for ten or twenty times as many pixels as are visible in the final picture. This

means the computational cost of any per-pixel operation—such as Phong shading or

15   texture-mapping—is multiplied by ten or twenty. The number of surfaces per pixel,

averaged over an entire frame, is called the depth complexity of the frame. In

conventional z-buffered renderers, the depth complexity is a measure of the

renderer's inefficiency when rendering a particular frame.

20   In accordance with the present invention, in a pipeline that performs deferred

shading, hidden surface removal (HSR) is completed before any pixel coloring is

done. The objective of a deferred shading pipeline is to generate pixel colors for

only those primitives that appear in the final image (i.e., exact HSR). Deferred

shading generally requires the primitives to be accumulated before HSR can begin.

25   For a frame with only opaque primitives, the HSR process determines the single

visible primitive at each sample within all the pixels. Once the visible primitive is

determined for a sample, then the primitive's color at that sample location is

determined. Additional efficiency can be achieved by determining a single per-pixel

color for all the samples within the same pixel, rather than computing per-sample

30   colors.

- 11 -

For a frame with at least some alpha blending (as defined in the above referenced OpenGL specification) of primitives (generally due to transparency), there are some samples that are colored by two or more primitives. This means the HSR process must determine a set of visible primitives per sample.

In some APIs, such as OpenGL, the HSR process can be complicated by other operations (that is by operation other than depth test) that can discard primitives. These other operations include: pixel ownership test, scissor test, alpha test, color test, and stencil test (as described elsewhere in this specification). Some of these operations discard a primitive based on its color (such as alpha test), which is not determined in a deferred shading pipeline until after the HSR process (this is because alpha values are often generated by the texturing process, included in pixel fragment coloring). For example, a primitive that would normally obscure a more distant primitive (generally at a greater z-value) can be discarded by alpha test, thereby causing it to not obscure the more distant primitive. A HSR process that does not take alpha test into account could mistakenly discard the more distant primitive. Hence, there may be an inconsistency between deferred shading and alpha test (similarly, with color test and stencil test); that is, pixel coloring is postponed until after HSR, but HSR can depend on pixel colors. Simple solutions to this problem include: 1) eliminating non-depth-dependent tests from the API, such as alpha test, color test, and stencil test, but this potential solution might prevent existing programs from executing properly on the deferred shading pipeline; and 2) having the HSR process do some color generation, only when needed, but this potential solution would complicate the data flow considerably. Therefore, neither of these choices is
attractive. A third alternative, called conservative hidden surface removal (CHSR), is one of the important innovations provided by the inventive structure and method. CHSR is described in great detail in subsequent sections of the specification.

Another complication in many APIs is their ability to change the depth test. The standard way of thinking about 3D rendering assumes visible objects are closer

- 12 -

than obscured objects (i.e., at lesser z-values), and this is accomplished by selecting
a "less-than" depth test (i.e., an object is visible if its z-value is "less-than" other
geometry). However, most APIs support other depth tests such as: greater-than,
less-than, greater-than-or-equal-to, equal, less-than-or-equal-to, less-than, not-equal,
5   and the like algebraic, magnitude, and logical relationships. This essentially
"changes the rules" for what is visible. This complication is compounded by an API
allowing the application program to change the depth test within a frame. Different
geometry may be subject to drastically different rules for visibility. Hence, the time
order of primitives with different rendering rules must be taken into account. If they
10 are rendered in the order A, B, then C, primitive C will be the final visible surface.
However, if the primitives are rendered in the order C, B, then A, primitive A will
be the final visible surface. This illustrates how a deferred shading pipeline must
preserve the time ordering of primitives, and correct pipeline state (for example, the
depth test) must be associated with each primitive.

15

Deferred Shading Graphics Pipeline
Provisional U.S. patent application serial no. 60/097,336; filed August 20,
1998, describes various embodiments of novel deferred Shading Graphics Pipelines.
The present invention, and its various embodiments, is suitable for use as the
20 Texture Block in the various embodiments of that differed shading graphics pipeline,
or for use with other graphics
pipelines which do not use differed shading. Details of such graphics pipelines are
for convenience not described again herein.

25   ### Texture
The Texture Block of a graphics pipeline applies texture maps to the pixel
fragments. Texture maps are stored in Texture Memory, which is typically loaded
from the host computer's memory using the AGP interface. In one embodiment, a
single polygon can use up to eight textures, although alternative embodiments allow
30 any desired number of textures per polygon.

- 13 -

The inventive structure and method may advantageously make use of trilinear mapping of multiple layers (resolutions) of texture maps. Texture maps are stored in a Texture Memory which may generally comprise a single-buffered memory loaded from the host computer's memory using the AGP interface. In the

5   exemplary embodiment, a single polygon can use up to eight textures. Textures are MIP-mapped. That is, each texture comprises a series of texture maps at different levels of detail, each map representing the appearance of the texture at a given distance from the eye point. To produce a texture value for a given pixel fragment, the Texture Block performs tri-linear interpolation from the texture maps, to

10 approximate the correct level of detail. The Texture Block can, in conjunction with the Fragment Block, perform other interpolation methods, such as anisotropic interpolation.

The Texture Block supplies interpolated texture values (generally as RGBA

15 color values) to the graphics pipeline shading block on a per-fragment basis. Bump maps represent a special kind of texture map. Instead of a color, each texel of a bump map contains a height field gradient. The multiple layers are MIP layers, and interpolation is within and between the MIP layers. The first interpolation is within each layer, then you interpolate between the two adjacent layers, one nominally

20 having resolution greater than required and the other layer having less resolution than required, so that it is done three-dimensionally to generate an optimum resolution.

Detailed Description of Texture Pipeline

25 Referring to Fig. 2, there is shown a block diagram of one embodiment of a texture pipeline constructed in accordance with the present invention. Texture unit 1200 receives texture coordinates for individual fragments, accesses the appropriate texture maps stored in

texture memory, and generates a texture value for each fragment. The texture

30 values are sent downstream, for example to a shading block which may then combine the texture value with other image information such as lighting to generate

the final color value for a fragment.

Texture Setup 1211 receives data packets, for example, from the Fragment
unit of U.S. Provisional Patent application 60/097,336. Data packets provide
5   texture LOD data for the texture maps, and potentially visible fragment data for an
image to be rendered. The fragment data includes (s, t, r) texture coordinates for
each fragment. As shown in Fig. 3, the (s, t) coordinates are normalized texture
space coordinates. For 3D textures, the "r" index is used to indicate texture depth.
The s and t coordinates are floating point numbers. Texture setup 1211 translates
10   the s, and t coordinates into i0, i1, j0, j1 (4 bilinear samples) and

LODA/LODB (adjacent LODs for trilinear mipmapping) coordinates. The i0, i1,
j0, j1 coordinates are 12 bit unsigned integers. LODA and LODB are 4 bit integers,
for example with LODA being the stored LOD greater than the actual LOD, and
15   LODB being the stored LOD less than the actual LOD. For 3D textures the r
coordinate is converted into a k coordinate. In a trilinear mipmapping embodiment,
each fragment has eight texture
coordinates associated with it. The i, j, and LOD/k values are all transferred to
Dualoct Bank Mapping unit 1212.

20

The Fragment Unit receives S, T, R coordinates in floating point format.
Setup converts these S, T, R coordinates into U, V, W coordinates, which are fixed
point coordinates used prior to texture look-up. The Texture Block then performs a
texture look-up and provides i, j, k coordinates, which are integer coordinates
25   mapped in normalized space. Thus, $u = i \times$ texture width, $v = j \times$ texture height,
and $w = k \times$ texture depth.

Texture Maps

Texture maps are allocated to Texture Memory 1213 and Texel Prefetch
30   Buffer 1216 using methods to minimize memory conflicts and maximize throughput.
Dualoct Bank Mapping unit 1212 maps the i, j, and LOD/k coordinates into Texture

- 15 -

Memory 1213 and Texel Prefetch Buffer 1216. Dualoct Bank Mapping unit 1212

also generates tags for texels stored in Texel Prefetch Buffer 1216. The tags are

stored in the eight Tag Banks 1216-0 through 1216-7. The tags indicate whether a

texel is stored in Texel Prefetch Buffer 1216, and the location of the texel in the

5   buffer.

Texture Memory Management Unit (MMU) 1210 controls access to Texture

Memory 1213. Texture Memory 1213 stores the active texture maps. If a texel is

not found in Texel

10

Prefetch Buffer 1216, then Texture MMU 1210 requests the texel from Texture

Memory 1213. If the texel is from a texture map not stored in Texture Memory

1213 then the texture map can be retrieved from another source as is shown in Fig.

2. Texture memory has, in various embodiments, access to Frame buffer 1221,

15 AGP memory 1222, Virtual memory 1223, with Virtual memory in turn having

access to disk 1224 and network 1225. Thus, a variety of locations are available for

texture addresses to be received in the event of a miss in order to greatly reduce the

instances where a needed texel is ultimately not available at the time it is needed in

the pipeline, since there is time between the determination of a texture cache miss

20 and the time that texel is actually needed later on down the pipeline.

After the texels for a given fragment are retrieved, Texture Interpolator 1218

interpolates the texel color values to generate a color value for the fragment. The

color value is then inserted into a packet and sent down the pipeline, for example to

25 a shading block.

A texture array is divided into 2x2 texel blocks. Each texel block in an array

is represented in Texture Memory. Texturing a given fragment with tri-linear

mipmapping requires accessing two to eight of these blocks, depending on where the

30 fragment falls relative to the 2x2 blocks. For trilinear mipmapping for each

fragment, up to eight texels must be retrieved from memory. Ideally all eight texels

- 16 -

are retrieved in parallel. As shown in Fig. 4a, to provide all eight texels in parallel, Texel Prefetch Buffer 1216 consists of eight independently accessible memory banks 1216-0 through 1216-7. Similarly, as shown in Fig. 5, Texture Memory 1213 includes a plurality of Texture Memory Devices, organized into a plurality of

5   channels, such as channels 1213-0 and 1213-1. To access all eight texels in parallel from Texel Prefetch Buffer 1216 each texel must be stored in a separate Prefetch Buffer Bank.

Texture Tile Addressing

10   To maximize the memory throughput the texels in the texture maps are re-mapped into a spatially coherent form using texture tile addresses. The texels required to generate adjacent fragments depend upon the orientation of the object being rendered, and the depth location of the object in the scene. For example, adjacent fragments of a surface of an object at a large skew angle with respect to the

15   viewing point will use texels at farther distances apart in the selected LOD than adjacent fragments of a surface that are approximately perpendicular to the viewing point. However, there is typically some spatial coherence between groups of fragments in close proximity and the texels used to generate texture for the fragments. Therefore, the texture tile addresses for the texels in the texture maps

20   are defined so as to maximize the spatial coherence of the texture maps.

        Figs. 6a and 6a illustrate a spatially coherent texel mapping for texture memory 1213, including texture map 800, including texture "super blocks" 800-0 through 800-3. In one embodiment, a RAMBUS™, RAMBUS Corp., Mountain

25   View California, memory is used for Texture Memory 1213. The smallest accessible data structure in RAMBUS memory is a "Dualoct" which is 16 bytes. Each texel contains 32 bits of color data in the format RGBA-8, or Lum/Alpha 16. Four texels can therefore be stored in each dualoct. The X and Y axis of Figs. 6a and 6b include dualoct labels. The (X,Y) coordinates correspond to the (i, j)

30   coordinates with the least significant bit of (i, j) dropped. Fig. 6a illustrates how the texels are renumbered within each dualoct. The texels are numbered sequentially

- 17 -

starting at the origin of each dualoct and increasing sequentially in a counterclockwise order. Fig. 6c shows how texel locations are remapped from linear addressing to a reconfigured address including a "swirl address" portion.

5

Referring to Fig. 6b, sector 800-0-0 shows the swirl pattern mapping for 16 dualocts. The four bit labels in each dualoct indicate the dualoct number that is used to generate an address for storing the dualoct in RAMBUS Texture Memory 1213 and Texel Prefetch Buffer 1216. Each dualoct shown in Fig. 6b contains 4 texels

10 arranged as shown in Fig. 6a. Dualocts are renumbered sequentially in groups of four, starting at the origin and moving in a counter-clockwise direction. After renumbering a group of dualocts, the next group of four dualocts are selected moving in a counter clockwise direction around the sector. After all four groups in a sector have been renumbered, the renumbering pattern is repeated for the next

15 sector (i.e., sector 800-0-1) moving counter-clockwise around a dualoct block. For example, after the 16 dualocts in sector 800-0-0, the dualoct numbers continue in sector 800-0-1 which contains dualoct numbers 16 - 30 which are numbered in the same pattern as sector 800-0-0.

This pattern is then repeated in sector 800-0-2 and in sector 800-0-3. Dualoct block

20 0 (800-0) consists of the four sectors 800-0-0 through 800-0-3. The dualoct block 0 pattern is then repeated in dualoct block 1 (800-1) starting with dualoct number 64, followed by dualoct block 2 (800-2), and dualoct block 3 (800-3). In one embodiment, the recursive swirl pattern stops at the texture super block 0 (800) level.

25

Alternative spatially coherent patterns are used in alternative embodiments, rather than the recursive swirl pattern illustrated in Figs. 6a and 6b. Fig. 7 illustrates a super block 900 of a texture map that is mapped using one such alternative pattern. Super block 900 includes sectors 0 - 15. The dualoct numbering

30 pattern within each sector is the same for the super block 900 pattern as for texture super block 0 (800) shown in Fig. 8. However, rather than repeating the counter-

- 18 -

clockwise swirl pattern at the sector level, the dualoct numbers at the sector level follow the pattern indicated by the sector numbers 0 - 15 in Fig. 7, limiting the swirl size to 64x64 texels.

5   Fig. 8 illustrates the dualoct numbering pattern at the super block level of a texture map 1000. At the super block level the pattern changes to a simple linear mapping, since in certain embodiments it has been determined that beyond 64x64 texels recursive swirling patterns begin to hurt spatial locality. The swirling is inherently a square operation, implying that it does not work very well at large sizes

10   of rectangular but non-square textures, and textures with border information. Limiting the swirl to 64x64 in certain embodiments of this invention, limits the minimum allocated size to a manageable amount of memory. In accordance with this invention, the swirling scheme provides that, upon servicing a miss request, the four samples fetched will reside in distinct

15   memory banks of the prefetched buffer, thus avoiding bank conflict. Furthermore, the swirling scheme maximizes subsequent hits to the prefetched buffer so that misses are typically spread out, so the memory system can service requests while the texture unit is working on hit data, thus minimizing stalls. The next super block of dualocts after texture super block 0 (800) is located directly to the right of texture

20   super block 0 (800). This linear pattern is repeated until super block n/64, and then a new row of super blocks is started with super block n/64+1, as shown.

       The spatially coherent texel mapping patterns illustrated in Figs. 8a, 8b and 9 are designed to maximize the likelihood that the four texels used to generate texture

25   for a fragment will be stored either in separate Texel Prefetch Buffer 1216 banks, or separate Texture Memory 1213 devices.

       ### Memory Addressing
       Referring to Fig. 4a, Texel Prefetch Buffer 1216 includes eight Prefetch

30   Buffer Banks 1216-0 through 1216-7. Fig. 4a shows how the numbered dualocts in Fig. 6b map into the eight Prefetch Buffer Banks 1216-0 through 1216-7. Also

shown are the four texels fetched

for a particular pixel location 899, shown in Fig. 6a, appearing without a memory
conflict. Fig. 4a shows the texels stored for one LOD. For trilinear mipmapping,

5   Banks 1216-4 through 1216-7 contain texels for the second LOD.

Referring to Fig. 5, there is shown a block diagram of one embodiment of
Texture Memory 1213. Texture Memory 1213 has two channels 1213-0 and 1213-
1. Each channel contains eight devices 1213-0-0 through 1213-0-7 and 1213-1-0

10 through 1213-1-7, respectively. Each device has an independent set of addresses
and independent I/O data lines to allow data to be independently accessed in each of
the eight devices. Each device contains
sixteen banks, meaning that in this embodiment there are 256 open pages, clearly
reducing the likelihood of memory conflict. In one embodiment each channel is a

15 64 Mbyte memory.

To map the texels in the texture map into a spatially coherent format,
Dualoct Bank Mapping unit 1212 generates a texture tile address for each dualoct.
Fig. 9 illustrates a texture tile address data structure 1180 according to one

20 embodiment of the present invention. Texture Field ID 1181 field is an 11 bit field
that defines the texture that is being referenced. Up to 2048 different textures can
be used in a single display. These textures may be stored in any memory resource.
Each fragment may then reference up to eight different textures. When a texture is
referenced that is not in Texture Prefetch Buffer 1216, Texture MMU 1210 loads

25 the memory from an external memory resource, and if necessary de-allocates the
required Texture Prefetch Buffer 1216 space to load the new texture. The LOD
1182 field is a 4 bit field that defines the LOD to be used in the selected texture
map. The U, V fields 1183 and 1184 are 11 bit fields for texture coordinates with a
range from 0 - 2047. The U, V fields for each dualoct are defined to generate the

30 spatially coherent format, such as the format

- 20 -

shown in Figs. 8a and 8b. For 3D textures, the 4 LSB's of the Texture field ID
1181 contain the 4 MSB's of the texture R coordinate, which is a texture depth index
generated from the k coordinate. Dualoct Bank Mapping unit 1212 provides the
four R coordinate bits whenever a 3D texture operation is in the pipeline.

5   Thereafter, 3D texture tile addresses are essentially treated the same as 2D and 1D
addresses.

         The texture tile address is provided to Texture MMU 1210 which generates a
corresponding texture memory address. Texture MMU 1210 performs the texture

10   tile address to texture memory address translation using a linear mapping of the
texture tile address into a table of texture memory addresses stored in Texture
Memory 1213. This table is maintained by software. Fig. 10 illustrates a texture
memory address data structure 1280 for a RAMBUS™ Texture Memory 1213.
Texture memory address data structure 1280 is designed to maximize the likelihood

15   that the dualocts required to generate the texture for a fragment will be stored in
different Texture Memory pages, as shown in Fig. 5. In one embodiment, Device
field 1285 consists of the least significant 3 bits of the texture memory address data
structure 1280. Device field 1285 defines the texture memory device that a dualoct
is stored in. Therefore, each sequential dualoct, as defined by the mapped texture,

20   is stored in a different texture memory device. The Bank field 1284 comprises the
next four low order bits, followed by a 1 bit Channel field 1283, a 9 bit Row field
1282 and a 6 bit Column field 1281.

         The texture memory address data structure 1280 is also programmable. This

25   allows the texture memory address to accommodate different memory
configurations, and to alter the placement of bit fields to optimize the access to the
texture data. For example, an alternative memory configuration may have more
than eight texture memory devices.

30   Texels are loaded from Texture Memory 1213 into Texel Prefetch Buffer
1216 to provide higher speed access. When texels are moved into Texel Prefetch

Buffer 1216, a corresponding tag is created in one of the eight Prefetch Buffer Tag

Blocks 1220-0 through 1220-7, shown in Fig. 4b. Each of the eight Tag Blocks

1220-0 through 1220-7 has a corresponding memory Queue 1230-0 through 1230-7.

Note that the tags are 64 entries, and the cache SRAM's are 256 entries. This

5   mapping allows each Prefetch Buffer tag entry to

map a "line" of 4 texels across four Prefetch Buffer Banks, as shown in Texel

Prefetch Buffer 1216 in Fig. 4a. This mapping allows 4 texels to be retrieved from

four separate Prefetch Buffer Banks every cycle, thus ensuring maximum texture

data access bandwidth. Each Tag Block may receive up to one texture tile address

10   per cycle. The texture tile address points to a particular dualoct of 4 texels. Each

Tag Block entry points to one dualoct line of texels in

Texel Prefetch Buffer 1216 memory. The incoming texture tile address is checked

against the contents of the Tag Block to determine whether the desired dualoct is

stored in Texel Prefetch Buffer 1216.

15

Fig. 4a shows the texels stored for one LOD. For trilinear mipmapping,

Banks 1216-4 through 1216-7 contain texels for the second LOD. The Texture ID

1181 bit [26] in the texture tile address is used to control whether an LOD gets

mapped to Prefetch Buffer Banks 0 - 3 (1216-0 through 1216-3) or Banks 4 - 7

20   (1216-4 through 1216-7). If Texture ID 1181 bit [26] = 0, then the even LOD's

(LOD[22] = 0) are mapped into Prefetch Buffer Banks 0 - 3, and the odd LOD's

(LOD[22] = 1) are mapped into Prefetch Buffer Banks 4 - 7. Conversely, if

Texture ID[26] = 1 then the odd LOD's are mapped into Prefetch Buffer Banks 0 -

3, and the even LOD's are mapped into Prefetch Buffer Banks 4 - 7. This mapping

25   ensures that all eight tags can be accessed in each cycle, and that texture information

is evenly distributed in

the caches. Dualoct Bank Mapping unit 1212 also follows this LOD mapping rule

when sending texture tile addresses to the corresponding Tag Block 1220-0 through

30   1220-7, shown in Fig. 4b.

To generate a texture for a fragment, Dualoct Bank Mapping unit 1212 generates up to eight dualoct requests, and sends them to the appropriate Prefetch Buffer Bank. The Prefetch Buffer Tags 1220-0 through 1220-7 are checked for a match. If there is a hit, the request is sent to the appropriate bank of Memory

5   Queue 1219. When the memory request exits Memory Queue 1219, the line number is sent to Texel Prefetch Buffer 1216 to look-up the data. If there is a miss on a given texture tile address, then a miss request is put into the miss queue for the corresponding tag block. The miss address is eventually read out of the miss queue and forwarded to Texture MMU 1210. The miss request is then serviced, the data is

10   retrieved from Texture Memory 1213 or another external memory source, and is ultimately provided to the appropriate Texel Prefetch Buffer Banks 1216-0 through 1216-7.

Each line in Memory Queue 1219 records one memory access for a particular

15   texture operation on one fragment of data. Memory requests are received at the top of Memory Queue 1219, and when they reach the bottom, Texel Prefetch Buffer 1216 is accessed for the data. Miss data is only filled into Texel Prefetch Buffer 1216 when a particular miss request reaches the bottom of the corresponding memory Queue 1230-0 through 1230-7.

20

Each of the eight memory Queues 1230-0 through 1230-7 hold up to eight pending miss addresses for a particular Prefetch Buffer Bank 1216-0 through 1216-7. If a memory Queue is not empty, then it can be assumed to contain at least one valid address. Every clock cycle Prefetch Buffer Controller 1218 scans the memory

25   Queues 1230-0 through 1230-7 searching for a valid entry. When a miss address is found, it is sent to Texture MMU 1210.

Fig. 9 is a Texture Tile Address Structure which serves as the tag for Texel Prefetch Buffer 1216. When this tag indicates a Texel Prefetch Buffer miss, a

30   Texture Memory 1213 look-up is needed. The Virtual Address Structure includes an 11 bit texture ID 1181, a four bit LOD 1182, and 11 bit U and V addresses 1183

- 23 -

and 1184. This Virtual Address of Fig. 9 serves as a tag entry in tag memories 1212-0 through 1212-7 (Fig. 2). In the event of a miss, a look-up in Texture Memory 1213 is required.

5   Fig. 10 depicts pointer look-up translation tag block 1190, which is stored, for example, in a dedicated portion of the texture memory, and is addressed using the 11 bit texture ID and four bit LOD number, forming a 15 bit index to locate the pointer of Fig. 10. The pointer, once located, points to a base address within texture memory where the start of the desired texture/LOD is stored. This base

10   address is then appended by addresses to be created by the U and V components of the virtual address to create the virtual address of a dualoct, which in turn is mapped to the physical address of RAMBus memory using the address structure of Fig. 11.

          Fig. 12 is a diagram depicting the address reconfigurations and process for

15   re-configuring the addresses with respect to Figs. 6c, 9, 10, and 12. As shown in Fig. 12, texture tile address structure 1180 (previously discussed with reference to Fig. 9) serves as a tag for Texel Prefetch Buffer 1216. When this tag indicates a Texel Prefetch Buffer miss, a texture memory 1213 look-up is needed. Translation buffer 1191 uses the 11-bit texture ID and four-bit LOD to form a 15 bit index to

20   pointer look-up translation tag block 1190 (previously discussed with reference to Fig. 10). Swirl addresses block 1192 remaps the bits from texture tile address data structure 1180 to form the "swirl address" 1194 (previously

25   discussed with respect to Figs. 6a-6c). Adder 1193 combines the pointer look-up translation tag block 1190 and "swirl address" 1194 to form the physical address 1280 to address RAMBus memory (as previously discussed with respect to Fig. 11).

Reorder Logic

30   Fig. 13a is a block diagram depicting one embodiment of Read Miss Control Circuitry 2600. Read Miss Control Circuitry 2600 receives a read miss request from

- 24 -

the miss logic shown in Fig. 2, when the tag mechanism determines that the desired information is not contained in texel prefetch buffer 1216. There are four types of read miss requests: texture look-up (miss), copy texture, read texture, and Auxring read dualoct (a maintenance utility function). The read miss requests received by

5   read control circuitry 2600 are prioritized by prioritization block 2620, for example, in the order listed above. Prioritization block 2620 sends the read request to the appropriate channel based upon the channel bit (Fig. 8) contained in the texture memory address to be accessed. These addresses are thus sent to request queues 2621-0 and 2621-1, which, in one embodiment, are 32 addresses deep. The

10   addresses stored in request queues 2621-0 and 2621-1 are applied to reorder logic circuity 2623-0 and 2623-1, respectively, which in turn access RAMBus memory controller 2649. Reorder logic 2623-0 and 2623-1 reorder the addresses received from request queues 2621-0 and 2621-1 in order to avoid memory conflict in texture memory, as will be described with respect to Fig. 13b. Since reorder logic 2623-0

15   and 2326-1 reorder the memory addresses to be accessed by RAMBus memory controller 2649, tag queue 2622 keeps track of channel and requester information. The accessed data is output to in-order return queue 2624, where the results are placed in the appropriate slots based upon the original order as indicated by queues 2609 and 2610. The data, once stored in proper order in in-order return queue 2624

20   is then provided to its requestor as data and a data valid signal. In one embodiment, the data is output in a 144 bits wide, which corresponds to a dualoct.

Fig. 13b is a block diagram of one embodiment of this invention which includes reorder logic 2623-0 (with reorder logic 2623-1 being identical), and

25   showing RAMBus memory controller 2649. The purpose of reorder logic 2623 is to monitor incoming address requests and reorder those requests so as to avoid memory conflicts in RAMBus memory controller 2649. For each memory address received as a request on Bus 2601, conflict detection block 2602 determines if a memory conflict is likely to occur based upon the addresses contained in first level reorder

30   queue 2603. If not, that address is directly forwarded to control block 2605, and is added to first level reorder queue 2603, to allow for conflict checking of

subsequently received addresses. On the other hand if a conflict is determined by conflict detection block 2602, the conflicting address request is sent to conflict queue 2604. In one embodiment, in order to prevent conflicting address requests from being utilized too distant from other requests received in the same recent time frame,

5   32 address requests are received by conflict detection block 2602 and either forwarded to control block 2605 (no conflict), or placed in conflict queue 2604, after which the addresses stored in conflict queue 2604 are output to control circuit 2605. In this manner, the reordered address requests are applied to reordered address queue 2606 to access RAMBus memory controller 2649 with

10   fewer, and often times zero, conflicts, in contrast to the conflict situations which would exist if the original order of the read request were applied directly to RAMBus memory controller 2649 without any reordering.

        In-Order tag queue 2609 and out-of-order tag queue 2610 maintains tag

15   information in order to preserve the original address order so that when the results are looked up and output from reorder logic 2623-0 and 2623-1, the desired (original) order is maintained.

        Information read from RAMBus memory controller 2649 is stored in read

20   data queue 2611. Through control block 2612, data from queue 2611 is forwarded to either out-of-order queue 2613 or in-order queue 2614. Control block 2615 reassembles data from queues 2613
and 2614 in the original request order and forwards it to the appropriate channel port of block 2614 in order. Control block 2624 receives channel specific data from

25   blocks 2623-0 and 2623-1 which is then re-associated and issued back to the waiting requester.

        The inventive pipeline includes a texture memory which includes a prefetch buffer. The host also includes storage for texture, which may typically be very

30   large, but in order to render a texture, it must be loaded into texture memory. Associated with each VSP are S and T's. In order to perform trilinear MIP

mapping, we necessarily blend eight (8) samples, so the inventive structure provides a set of eight content addressable (memory) caches running in parallel. In one embodiment, the cache identifier is one of the content addressable tags, and that's the reason the tag part of the cache and the data part of the cache are located

5   separate. Conventionally, the tag and data are co-located so that a query on the tag gives the data.. In the inventive structure and method, the tags and data are split up and indices are sent down the pipeline.

The data and tags are stored in different blocks and the content addressable

10   look-up is a look-up or query of an address, and even the "data" stored at that address in itself an index that references the actual data which is stored in a different block. The indices are determined, and sent down the pipeline so that the data referenced by the index can be determined. In other words, the tag is in one location, the texture data is in a second location, and the indices provide a link

15   between the two storage structures.

In one embodiment of the invention, the prefetch buffer comprises a multiplicity of associative memories, generally located on the same integrated circuit as the texel interpolator. In the preferred embodiment, the texel reuse detection

20   method is performed in the Texture Block.

In conventional 3-D graphics pipelines, an object in some orientation in space is rendered. The object has a texture map associated with it, which is represented by many triangle primitives. The procedure implemented in software, will instruct the

25   hardware to load the particular object texture into a Texture Memory. Then all of the triangles that are common to the particular object and therefore have the same texture map are fed into the unit and texture interpolation is performed to generate all of the colored pixels needed to represent that particular object. When that object has been colored, the texture map in DRAM can be destroyed since, for example by

30   a reallocation algorithm, the object has been rendered. If there are more than one object that have the same texture map, such as a plurality of identical objects

(possibly at different orientations or locations), then all of that type of object may desirably be textured before the texture map in DRAM is discarded. Different geometry may be fed in, but the same texture map could be used for all, thereby eliminating any need to repeatedly retrieve the texture map from host memory and

5   place it temporarily in one or more pipeline structures.

In more sophisticated conventional schemes, more than one texture map may be retrieved and stored in the memory, for example two or several maps may be stored depending on the available memory, the size of the texture maps, the need to

10   store or retain multiple texture maps, and the sophistication of the management scheme. Each of these conventional

texture mapping schemes, spatial object coherence is of primary importance. At least for an entire single object, and typically for groups of objects using the same

15   texture map, all of the triangles making up the object are processed together. The phrase spatial coherency is applied to such a scheme because the triangles form the object and are connected in space, and therefore spatially coherent.

In the inventive structure and method, a sizable memory is supported on the

20   card. In one implementation 128 megabytes are provided, but more or fewer megabytes may be provided. For example, 32 Mb, 64 Mb, 256 Mb, 512 Mb, or more may be provided, depending upon the needs of the user, the real estate available on the card for memory, and the density of memory available.

25   Rather that reading the eight texels for every visible fragment, using them, and throwing them away so that the eight texels for the next fragment can be retrieved and stored, the inventive structure and method stores and reuses them when there is a reasonable chance they will be needed again.

30   It would be impractical to read and throw away the eight texels every time a visible fragment is received. Rather, it is desirable to make reuse of these texels,

because if you're marching along in tile space, your pixel grid within the tile (typically processed along sequential rows in the rectangular tile pixel grid) could come such that while the same texture map is not needed for sequential pixels, the same texture map might be needed for several pixels clustered in an area of the tile,

5   and hence needed only a few process steps after the first use. Desirably, the invention uses the texels that have been read over and over, so when we

need one, we read it, and we know that chances are good that once we have seen one fragment requiring a particular texture map, chances are good that for some

10   period of time afterward while we are in the same tile, we will encounter another fragment from the same object that will need the same texture. So we save those things in this cache, and then on the fly we look-up from the cache (texture reuse register) which ones we need. If there is a cache miss, for example, when a fragment and texture map are encountered for the first time, that texture map is

15   retrieved and stored in the cache.

Texture Map retrieval latency is another concern, but is handled through the use of First-In-First-Out (FIFO) data structures and a look-ahead or predictive retrieval procedure. The FIFO's are large and work in association with the CAM.

20   When an item is needed, a determination is made as to whether it is already stored, and a designator is also placed in the FIFO so that if there is a cache miss, it is still possible to go out to the relatively slow memory to retrieve the information and store it. In either event, that is if the data was in the cache or it was retrieved from the host memory, it is placed in the unit memory (and also into the cache if newly

25   retrieved).

Effectively, the FIFO acts as a sort of delay so that once the need for the texture is identified (prior to its actual use) the data can be retrieved and re-associated, before it is needed, such that the retrieval does not typically slow down

30   the processing. The FIFO queues provide and take up the slack in the pipeline so that it always predicts and looks ahead. By examining the FIFO, non-cached texture

- 29 -

can be identified, retrieved from host memory, placed in the cache and in a special unit memory, so that it is ready for use when a read is executed.

5   The FIFO and other structures that provide the look-ahead and predictive retrieval are provided in some sense to get around the problem created when the spatial object coherence typically used in per-object processing is lost in our per-tile processing. One also notes that the inventive structure and method makes use of any spatial coherence within an object, so that if all the pixels in one object are done sequentially, the invention does take advantage of the fact that there's temporal and

10   spatial coherence.

The Texture Block caches texels to get local reuse. Texture maps are stored in texture memory in 2x2 blocks of RGBA data (16 bytes per block) except for normal vectors, which may be stored in 18 byte blocks.

15

Virtual Texture Numbers

The user provides a texture number when the texture is passed from user · space with OpenGL calls. The user can send some triangles to be textured with one map and then change the texture data associated with the same texture number to

20   texture other triangles in the same frame. Our pipeline requires that all sets of texture data for a frame be available to the Texture Block. The driver assigns a virtual texture number to each texture map.

Texture Memory

25   Texture Memory stores texture arrays that the Texture Block is currently using. Software manages the texture memory, copying texture arrays from host memory into Texture Memory. It also maintains a table of texture array addresses in Texture Memory.

30

- 30 -

Texture Addressing

The Texture Block identifies texture arrays by virtual texture number and LOD. The arrays for the highest LODs are lumped into a single record. A texture array pointer table associates a texture array ID (virtual texture number concatenated with the LOD) with an address in Texture Memory. We need to support thousands of texture array pointers, so the texture array pointer table will have to be stored in Texture Memory. We need to map texture array IDs to addresses approximately 500M times per second. Fortunately, adjacent fragments will usually share the same the texture array, so we should get good hit rates with a cache for the texture array pointers. (In one embodiment, the size of the texture array cache is 128 entries, but other sizes, larger or smaller, may be implemented.)

The Texture Block implements a direct map algorithm to search the pointer table in memory. Software manages the texture array pointer table, using the hardware look-up scheme to store table elements.

Texture Memory Allocation

Software handles allocation of texture memory. The Texture Block sends an interrupt to the host when it needs a texture array that is not already in texture memory. The host copies the texture array from main memory frame buffer to texture memory, and updates the texture array pointer table, as described above. The host controls which texture arrays are overwritten by new data.

The host will need to rearrange texture memory to do garbage collection, etc. The hardware will support the following memory copies:

- host to memory
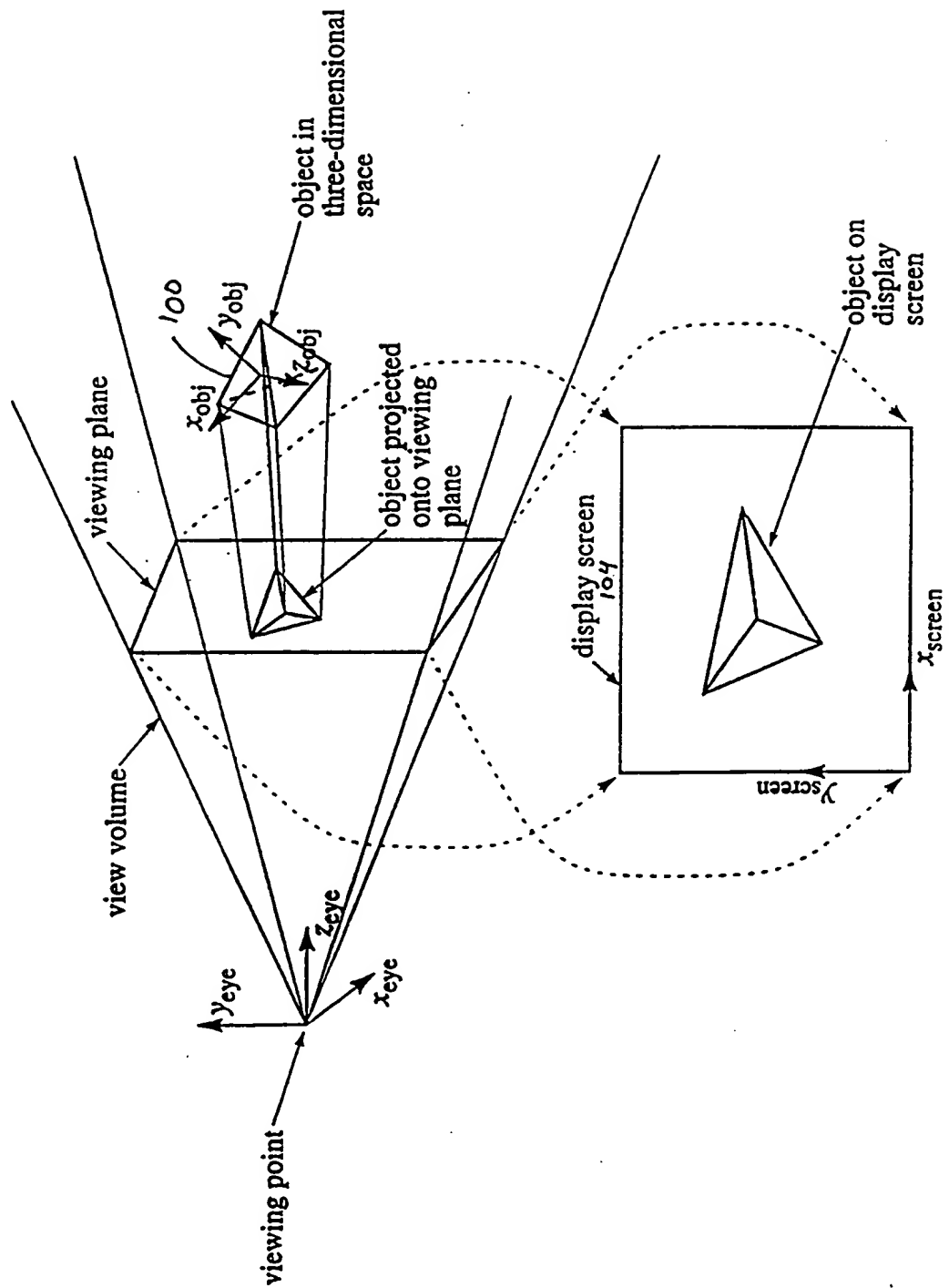- memory to host
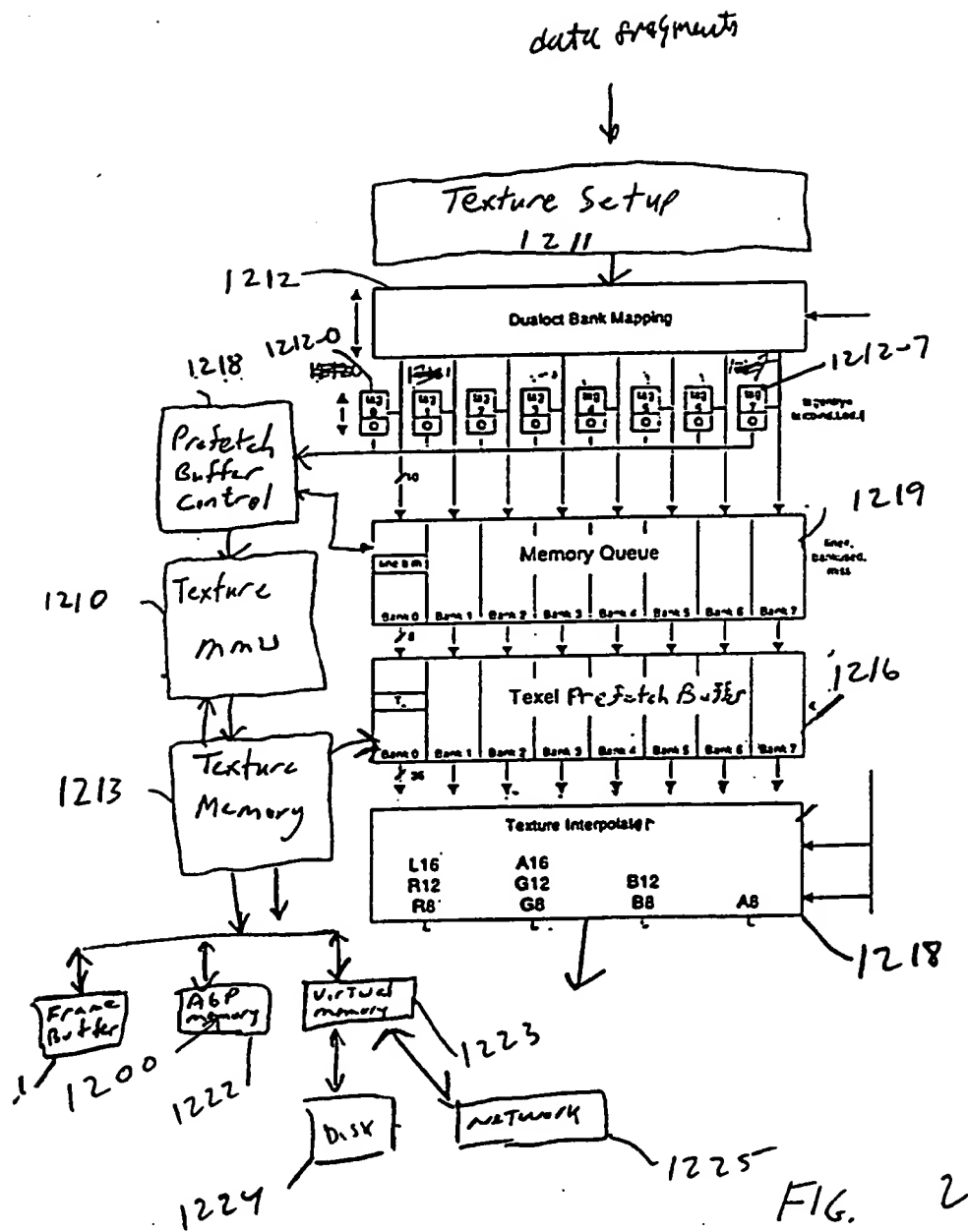- memory to memory

Alternative Embodiments

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those

5   skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A deferred graphics pipeline processor comprising:

a texture unit and a texture memory associated with said texture unit; said

5   texture unit applying texture maps stored in said texture memory, to pixel

fragments; said textures being MIP-mapped and comprising a series of texture maps

at different levels of detail, each map representing the appearance of the texture at a

given distance from an eye point; said texture unit performing tri-linear interpolation

from said texture maps to produce a texture value for a given pixel fragment that

10   approximates the correct level of detail; said texture memory having texture data

stored and accessed in a manner which reduces memory access conflicts and thus
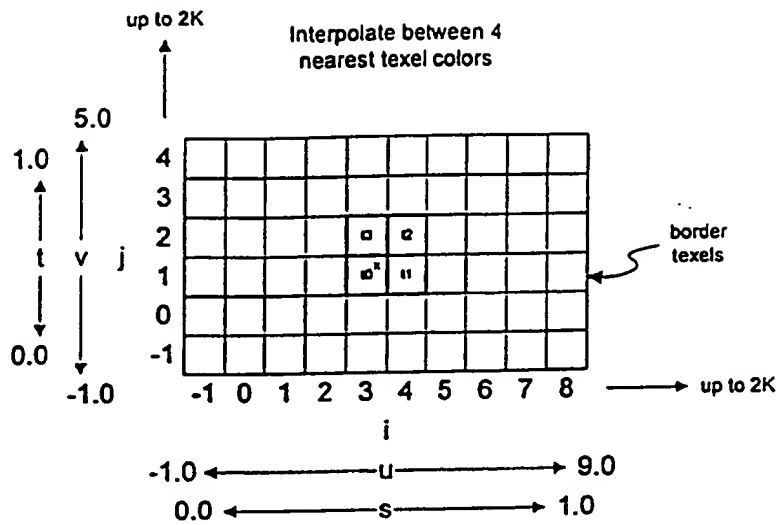
improves throughput of said texture unit.
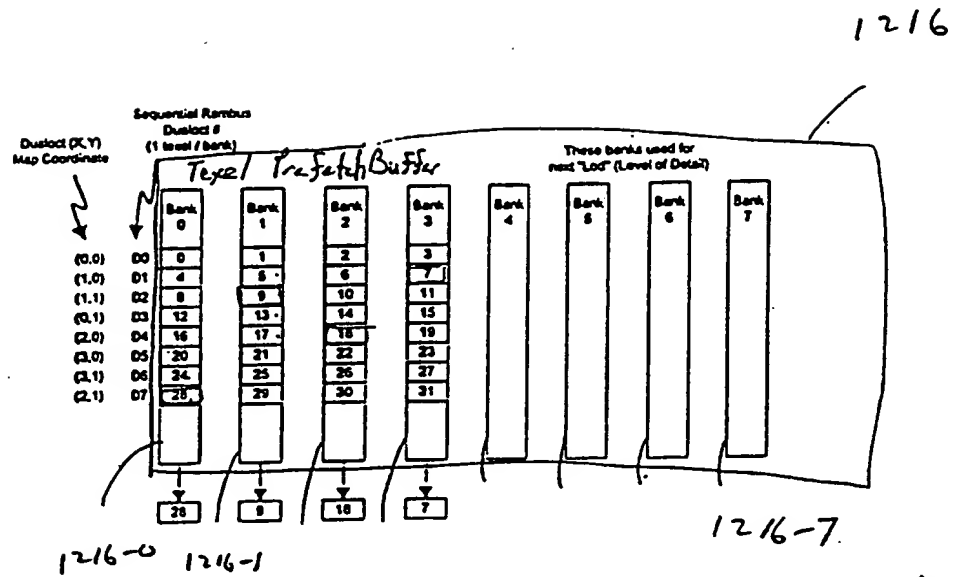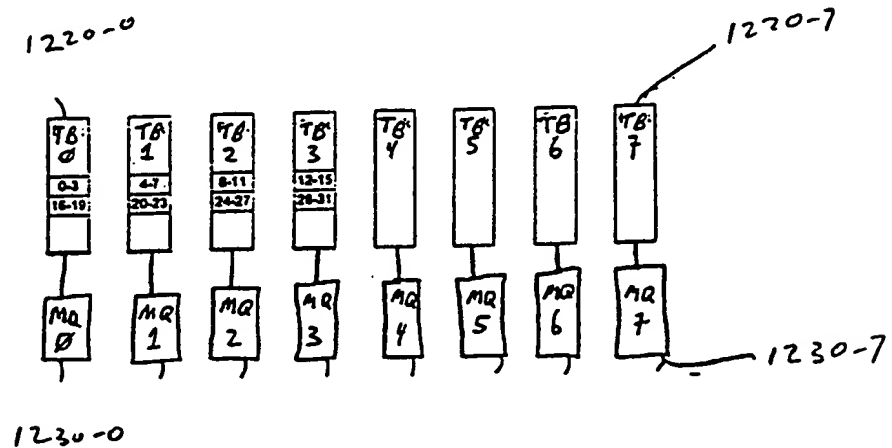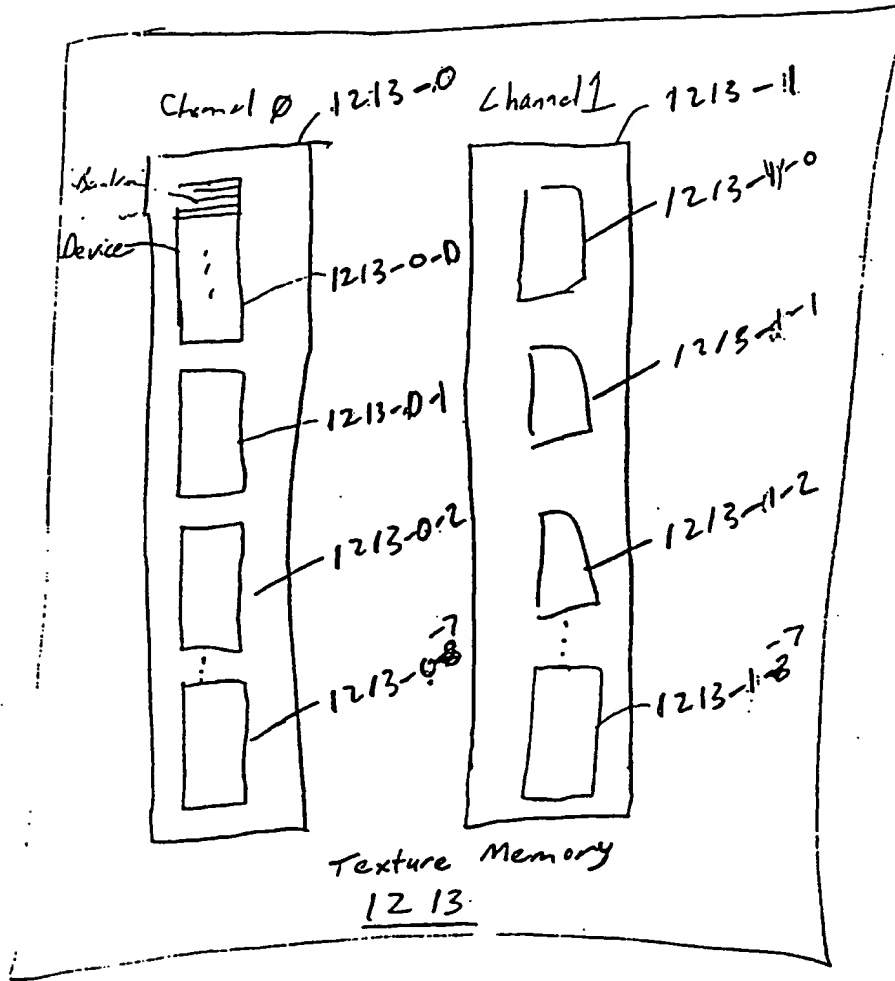
FIGURE 1

FIG. 2

FIG. 3

FIG. 4A

FIG. 4B

FIG.   5

800

2
800-2

3
800-3

i_0,j_0 i_1,j_0 800-0

Y - dustod

3
800-0-3

800-0-2
2

1 "Tile"
of 64 texels

Y - dustod

62 63	58 59	46 47	42 43
60 61	56 57	44 45	40 41
50 51	54 55	34 35	38 39
48 49	52 53	32 33	36 37
14 15	10 11	30 31	26 27
12 13	8 9	28 29	24 25
2 3	6 7	18 19	22 23
0 1	4 5	16 17	20 21

1111 15	1110 14	1011 11	1010 10
1100 12	1101 13	1000 8	1001 9
0011 3	0010 2	0111 7	0110 6
0000 0	0001 1	0100 4	0101 5

1
800-1

0 1 2 3 4 5 6 7
0   1   2   3   X - dustod

000 001 010 011 100 101 110 111   X - dustod

111
110
101
100
011
010
001
000

i_0,j_1

i_1,j_1

899

FIG. 6A

800-0-0

800-0-1 FIG.   6B

GENERATE UPTO 4 MISSES (DUALOCS)

Tex ID	LOD	y	U

Tex ID	LOD	V	U	V	U

"linear address"   "swirl address"

Fig 6c

FIG. 7

1000

m texels high

Y - direction

| n/64 + 1 | n/64 + 2 | n/64 + 3 | • • • • |
| 0 | 1 | 2 | • • • • n/64 |

1 "super block"
of 64 x 64 texels
(swirl pattern inside)

800   800   800   X - direction

n texels wide

FIG.   8

1180

Virtual Address

TEX ID (11)	LOD (4)	u (11)	V (11)

1181   1182   1183   1184

Fig 9

Valid Bit	pointer to start of LOD
16 J	24 bits

1190

Fig 10

1280

Texture Memory Address

Col. (6)	Row (9)	CH (1)	Bank (4)	Device (3)

1281   1282   1283   1284   1285

FIG. 11

Fig 9

Fig 10

Fig 6c

Fig 11

Fig 12

RdCtl

Rd Req (4 sources)

Prioritize
Gen Xsaction Pkts, Determine Ch
Gen Tags
Other Control

2620

2621-1

2621-0   RegQ
Ch0
(32)

RegQ
Ch1
(32)

Tag
Q
(128)

2622

Data   DataAvail

144

Sel Channel
Route to Correct Reg
Other Control

2624

ReOrder
logic
Ch0

2623-0

ReOrder
logic
Ch1

2623-1

↖ 2600

RMC

2649

Fig 13a

Fig 13 b